

Outspect: Live Memory Forensic & Incident Response for Virtual Machine

Syscan Singapore, 2-3/7/2009

Nguyen Anh Quynh, Kuniyasu Suzuki, Ruo Ando



Who am I?

- Working for a research institute in Japan.
 - National Institute of Advanced Industrial Science & Technology (AIST), Japan
- A member of Vnsecurity.net
- Interests: Operating System, Virtualization, Trusted computing, IDS, malware, digital forensic, ...

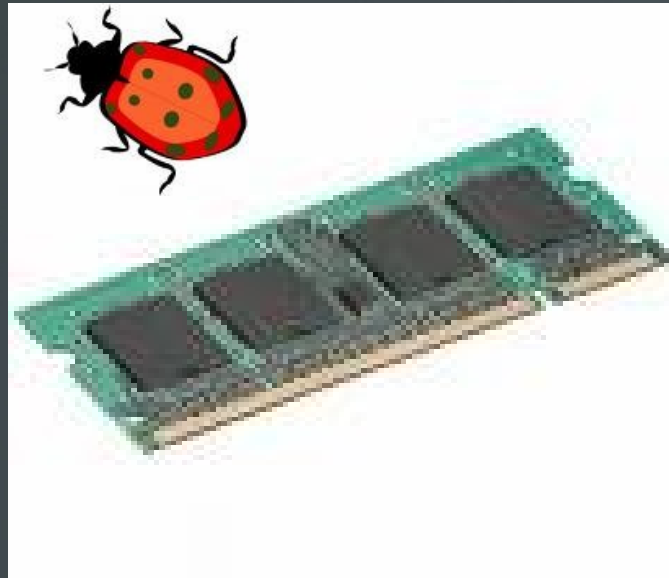


Agenda

- Live memory forensic problems
- Live memory forensic for Virtual Machine (VM)
- **Outspect** architecture/design/implementation
 - Focus on **Xen** (as hypervisor) and **Windows** (as guest)
- Demo on detecting malware with **Outspect**
- Conclusions



Live memory forensic problems



Digital forensic & incident response

- Despite a lot of defense layers, finally systems still got hacked!
 - Accept the fact, and ready to perform incident response when bad thing happen
 - Data forensic: Try to understand the intrusion
 - Trying to fix/recover compromised system if possible

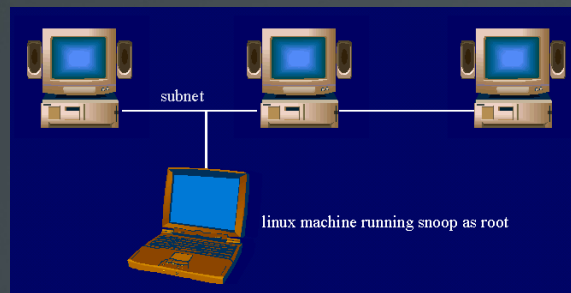


Forensic & incident response tools

- Inspect compromised system to answer various questions after the system is exploited
 - When?
 - What?
 - Who?
 - Why?
- Collect information as evidence
 - Can be used for law enforcement if requested
- Fix and recover compromised system, if possible.

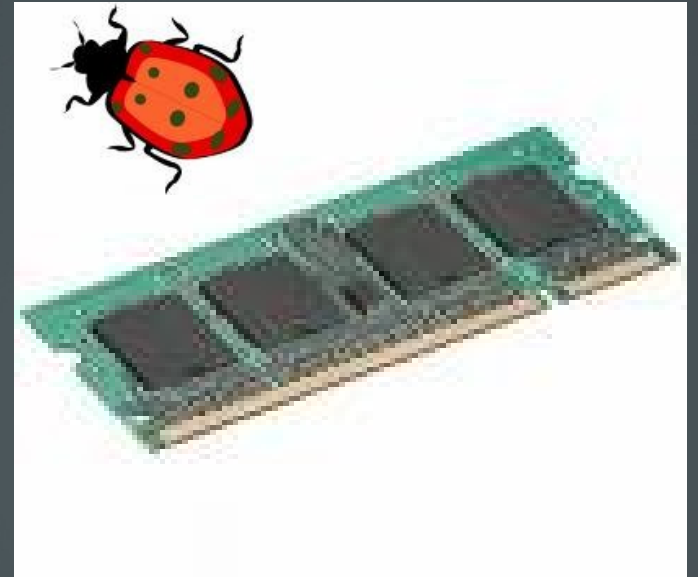
Forensic targets

- Hard-disk forensic
- Network forensic
- Memory forensic



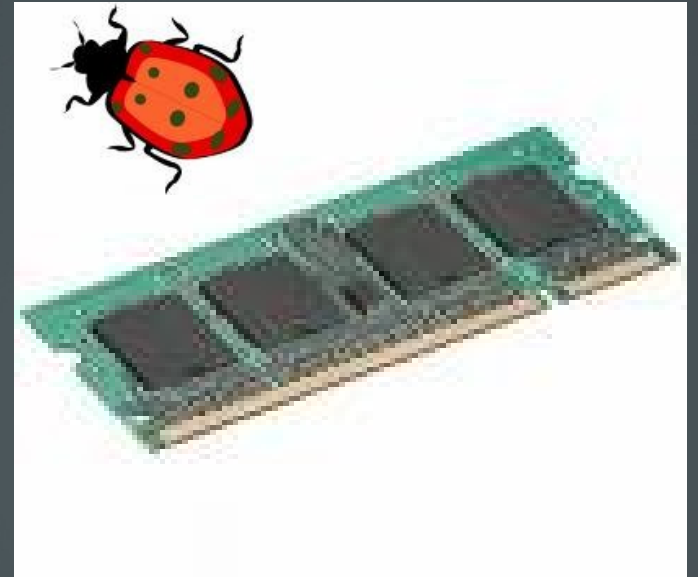
Memory forensic

- Advanced malware only exists in memory, but never write down to disk
 - Effective method to evade malware scanner
- Off-line forensic
- Online forensic



Live memory forensic tools

- Tool to inspect live systems
 - Capture live memory (for offline inspection)
 - Analyze live memory
 - Extract out system objects to understand what is happening
 - Find evidences of intrusion



Problems of live memory forensic

- Erase evidences in the memory
- Inconsistency memory problem
- Captured data process can be easily tampered by existent malware
 - Kernel malware



I dream a dream ...

- A perfect forensic/incident response tool?
 - Never erase evidences in the memory
 - No consistency problem
 - Cannot be, or very hard to be, tampered by malware
 - Even if malware run at OS level

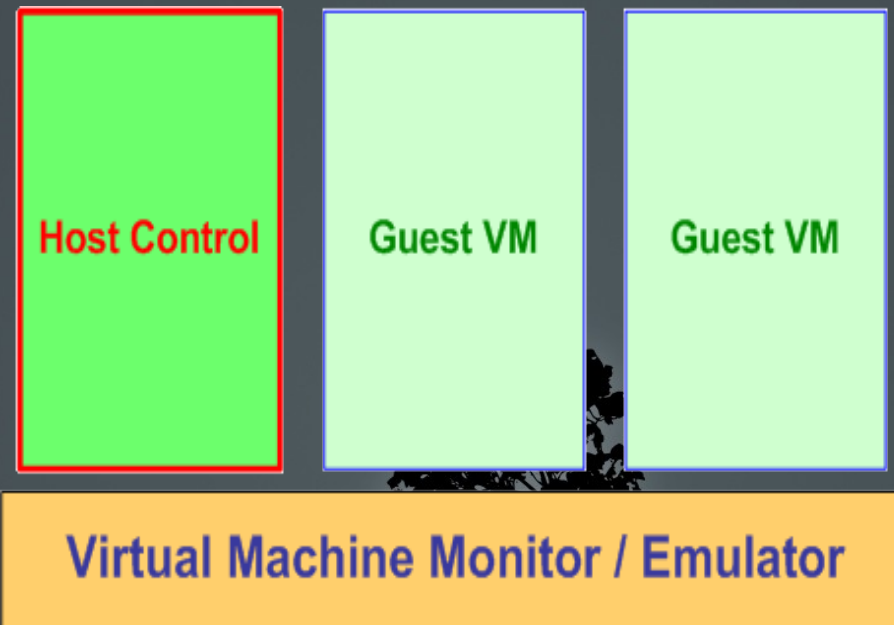


Outspect: live memory forensic & IR for Virtual Machine



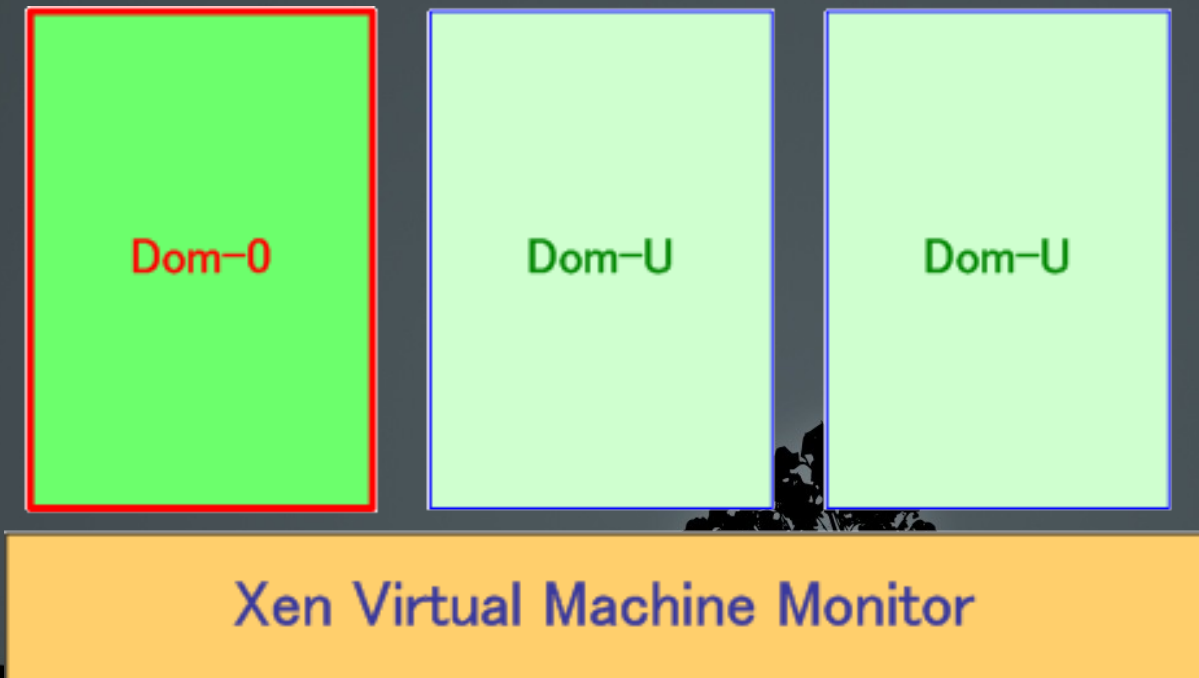
Virtual Machine Concept

- Running multiple virtual systems on a physical machine at the same time
- Multiple Operating Systems are supported
 - Windows, Linux, BSD, MacOSX, ...



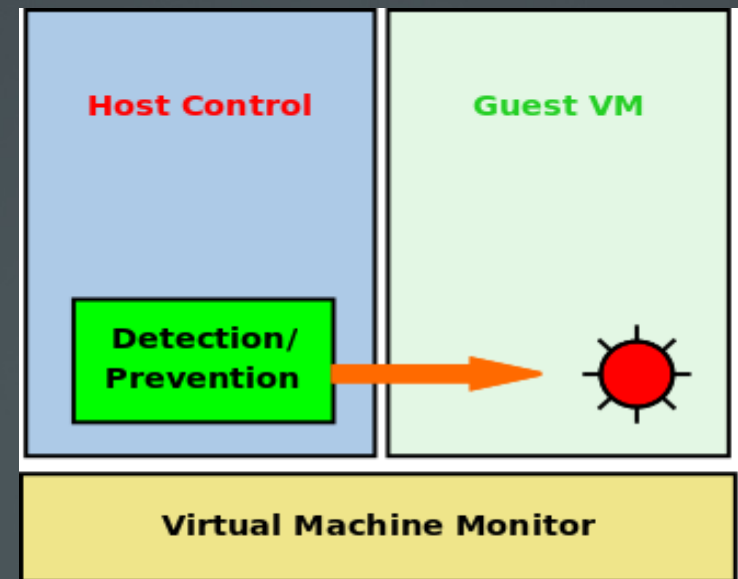
Virtual Machine: Xen

- Host: Dom0
- Guest: DomU
 - Paravirtualized guest
 - Full-virtualized guest (HVM)



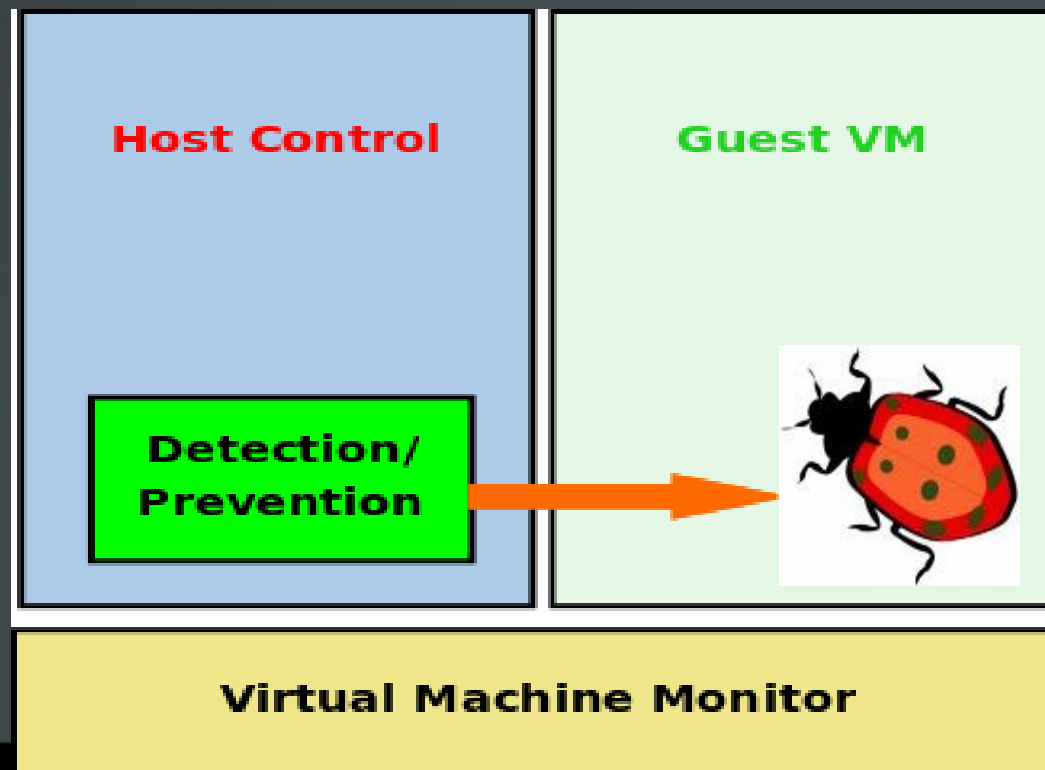
Approach

- Put the forensic and incident response tool outside of targeted VM
- Let it access to the VM memory to perform the job from outside
 - Scan memory to retrieve information
 - Can also manipulate memory (i.e write to) to disable malware, recover system if desired



Protect Virtual System

- Run the forensic/IR tools in the privileged VM
 - Access to protected VM thanks to VM interface
 - Focus on **Xen** in this talk



A dream comes true!

- Satisfy all the dreamed requirements, and even more
 - Never erase evidences
 - No more consistency memory problem
 - Pause VM before inspecting
 - Work on memory **snapshot**, but not "real" memory
 - Very hard to be tampered, or disabled by malware
 - Get the right information, even if malware run at Operating System level
 - Invisible to malware
 - Can effectively disable malware from outside



Challenges

- Analyzing raw memory to understand internal context of protected system

(1) Understanding virtual memory

- We have only physical memory access

(2) Retrieve OS-semantic objects

- Require excellent understandings on target OS internals

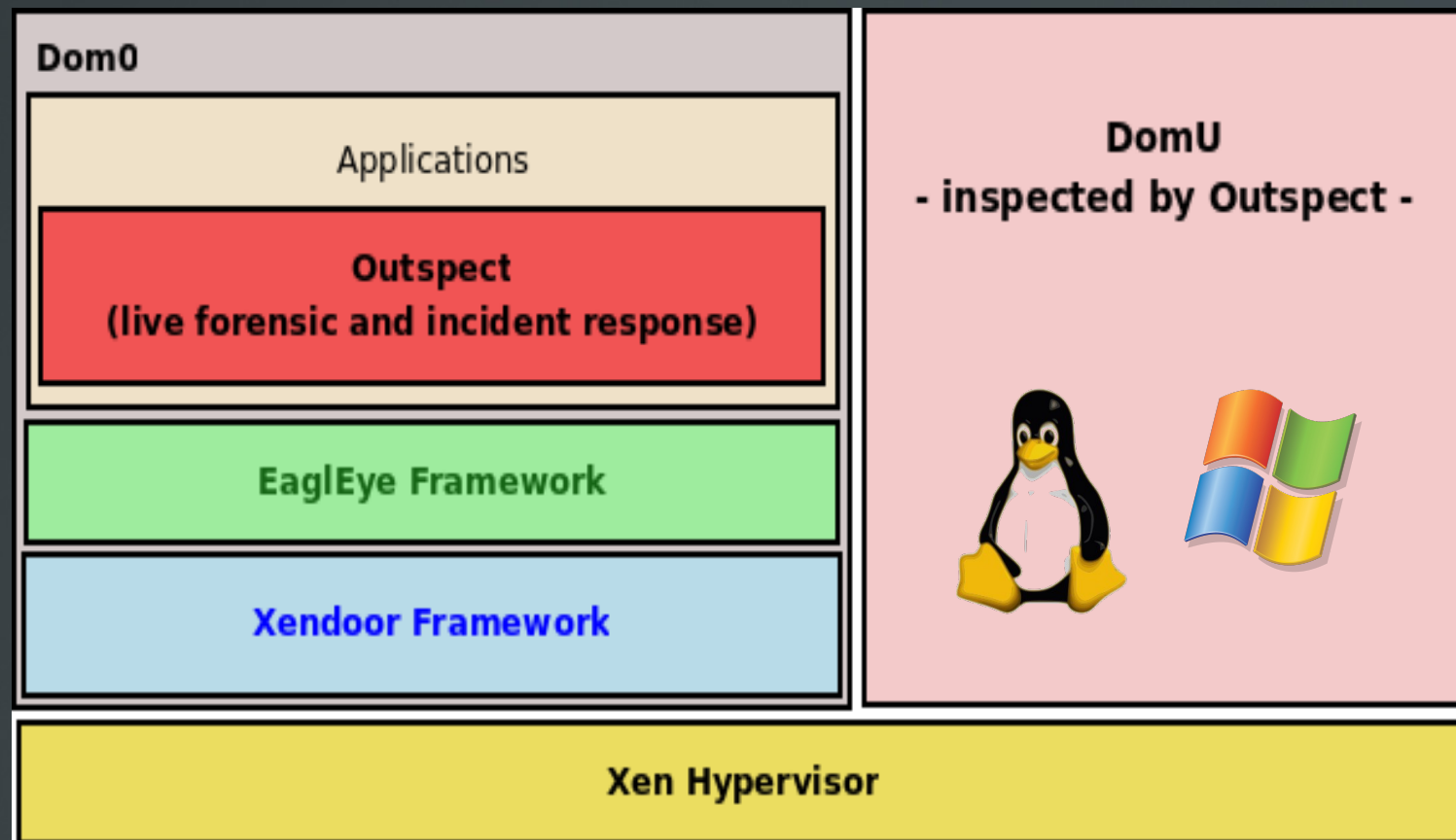


Multiple-layer Frameworks Architecture

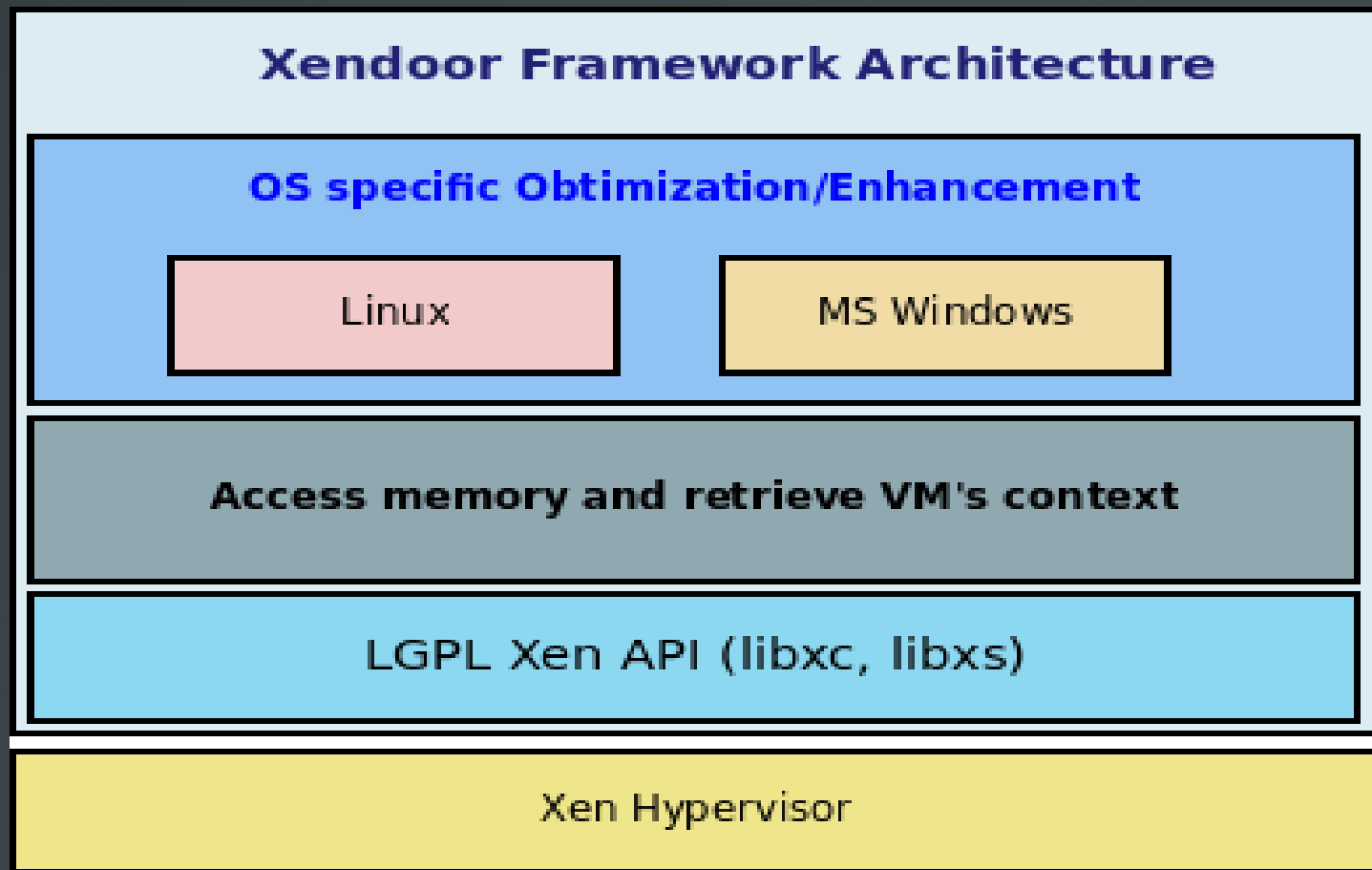
- Understanding virtual memory
 - **XenDoor** framework
- Retrieve OS-semantic objects
 - **EaglEye** framework



Full architecture

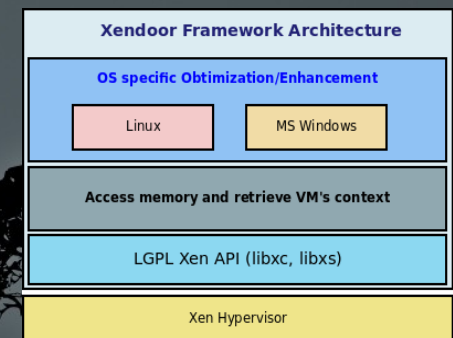


XenDoor framework



XenDoor framework

- Access to physical memory of protected system
 - OS independence
 - Provide access to virtual memory
 - Play a role of Memory-management-unit (MMU)
 - Software-based MMU
 - Must be able to understand all the memory mode (legacy/2MB pages/PAE,...)
- Provide access to registers of protected system



Sample XenDoor API

```
/* <xendoor/xendoor.h> */
```

```
/* Read data from memory of a process running inside a Xen domain. */
```

```
int xendoor_read_user(xendoor_t h, unsigned long pgd, unsigned long vaddr,  
                      void *buf, unsigned int size);
```

```
/* Write data into memory of a process running inside a Xen domain. */
```

```
int xendoor_write_user(xendoor_t h, unsigned long pgd, unsigned long addr,  
                      void *buf, unsigned int size);
```

```
/* Read data from a Xen domain's physical memory. */
```

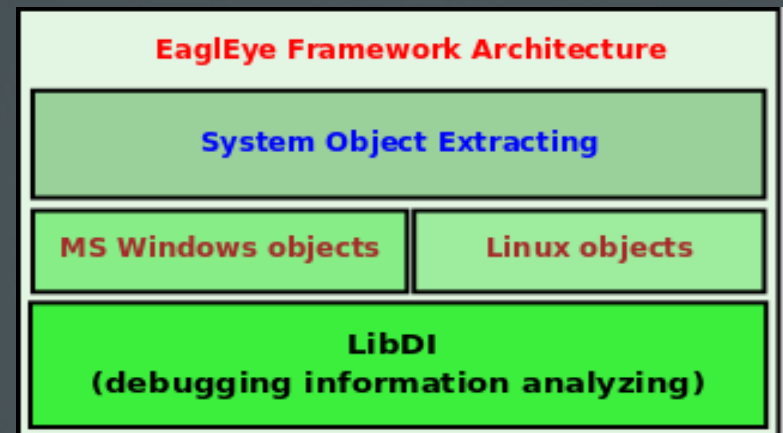
```
int xendoor_read_gpa(xendoor_t h, unsigned long paddr, void *buf, unsigned int  
size);
```

```
/* Write data into a Xen domain's physical memory. */
```

```
int xendoor_write_gpa(xendoor_t h, unsigned long paddr, void *buf, unsigned int  
size);
```


EaglEye Framework

- Use the service provided by **AnyDoor** to access to virtual/physical memory of protected system
- Retrieve OS-objects
- Focus on important objects, especially which usually **exploited by malware**, or can **disclose their residence**
 - Network ports, connections
 - Processes
 - Kernel modules
 -



Challenges

- Retrieve semantic objects requires excellent understanding on OS internals

(1) Locate the objects

- We have only physical memory access

(2) Actually retrieve objects and its internals

- How the objects are structured?
 - Structure size?
 - Structure members?
 - Member offset?
 - Member size?
 - Member attributes?



Locate OS's objects

- Kernel modules
- Processes/threads
- System handles
- Open files
- Registries
- DLLs
- Network connections/ports
- Drivers, symbolic links, ...



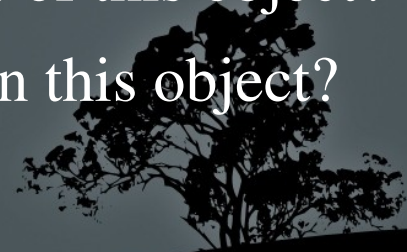
Retrieve objects

- Must understand object structure
 - Size, members, ...
 - Must be able to support various versions of Windows
- List the offset of every popular objects, with offsets of popular fields?
 - Does by everybody else
 - But this is far from good enough!



Another dream ...

- To be able to query structure of all the objects, with their fields
 - Support all kind of OS, with different versions
 - On demand, with all kind of objects
 - Various questions are possible
 - What is the size of this object?
 - What is the offset of this member field in this object?
 - What is the size of this array member of this object?
 - What is the position of this bit-field in this object?
 - ...



... comes true, again: LibDI

- Satisfy all the above requests, and make your dream come true
 - Come in a shape of another framework
 - Rely on public information on OS objects
 - OS independence
 - Windows and Linux are well supported so far
 - Have them in debugging formats DWARF , and extract their structure out at run-time



Windows objects

- **ReactOS** file header prototypes
 - Free & open to public
- Support Win2k3 and up.
- Always updated and under active development



Windows objects - problems

- **ReactOS** only supports Win2k3 and up
- Need to patch headers to support WinXP and prior versions
 - From Windows debugging symbols data
 - Patch size is small



Sample LibDI API

```
/* <libdi/di.h> */
```

```
/* Get the struct size, given its struct name */
```

```
int di_struct_size(di_t h, char *struct_name);
```

```
/* Get the size of a field of a struct, given names of struct and member. */
```

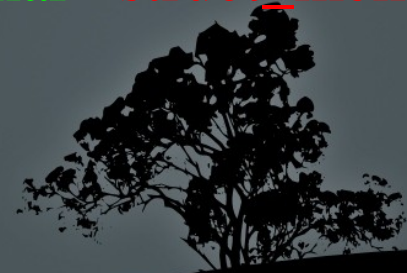
```
int di_member_size(di_t h, char *struct_name, char *struct_member);
```

```
/* Get the size of a bit field of a struct, given names of struct and member. */
```

```
int di_member_bit_size(di_t h, char *struct_name, char *struct_member);
```

```
/* Get the offset of a field member of a struct, given names of struct and member */
```

```
int di_member_offset(di_t h, char *struct_name, char *struct_member);
```



Retrieve objects

- Separate API for each kind of objects
- Designed so it is hard to be abused or tampered by guest VM
 - Get first object in the list of objects
 - Usually the head of object list must be located
 - Or by scanning the pool memory
 - Using pattern-matching technique
 - Get next objects
 - One by one, until done



Sample EagleEye API (1)

```
/* <eagleeye/eagleeye.h> */
```

```
/* @task: output value, pointed the the kernel memory keep task info */
```

```
int ee_get_task_first(ee_t h, unsigned long *task);
```

```
/* @task: output value, pointed the the kernel memory keep task info */
```

```
int ee_get_task_next(ee_t h, unsigned long *task);
```

```
/* get the pointer to the process struct, given the process's pid.
```

```
int ee_get_task_pid(ee_t h, unsigned long pid, unsigned long *task);
```

```
/* get the first open dll file of a task with a given process id.
```

```
 * on return, dll points to the userspace memory that keeps dll info */
```

```
int ee_get_task_dll_first(ee_t h, unsigned long pid, unsigned long *dll);
```

```
/* get the next open dll file of a task with a given process id.
```

```
int ee_get_task_dll_next(ee_t h, unsigned long *dll);
```

Sample EagleEye API (2)

```
/* <eagleeye/windows.h> */
```

```
/* get process image filename, given its EPROCESS address */
```

```
int windows_task_imagename(ee_t h, unsigned long eprocess, char *name,  
    unsigned int count);
```

```
/* get process id, given its EPROCESS address */
```

```
int windows_task_pid(ee_t h, unsigned long eprocess, unsigned long *pid);
```

```
/* get parent process id, given its EPROCESS address */
```

```
int windows_task_ppid(ee_t h, unsigned long eprocess, unsigned long  
    *ppid);
```

```
/* get process cmdline, given its EPROCESS address */
```

```
int windows_task_cmdline(ee_t h, unsigned long eprocess, char *cmdline,  
    unsigned int count);
```



EaglEye Architecture

EaglEye Framework Architecture

System Object Extracting

MS Windows objects

Linux objects

LibDI
(debugging information analyzing)

Outspect toolset

- A range of command-line tools to gather and investigate information
 - Ready to script them for automation information collection
- A shell to connect them all: **oshell**
 - Same syntax, same behaviour
 - Use the same code that is refactored out to work for both shell and separate commands
 - Convenient to use
 - System shell, System command
 - Redirect output to other commands, or pipe them to other system commands
 - Optimization for speed
 - Cache frequently-used data



Outspect toolset (2)

- Gather information that can prove the existence of malware
 - **pe**: PE file analyzing
 - **view**: View memory in hex/string format
 - **dump**: Dump memory out (physical or process or kernel)
 - **write**: Write to memory
 - **search**: Searching (pattern matching, regex, ...)
 - **ps/pstree**: Processes
 - **dlls**: DLLs, **registry**: Registries, **files**: Open files, **vad**: VADs
 - **kmod**: Kernel modules
 - **address**: Attributes of a memory address
 - **connection**: Open network connections, **socket**: open sockets
 - **disasm**: Disassemble memory range
 - **register**: Registers





Outspect toolset demo



Metasploit



Metasploit payloads

- Memory-resident only payloads
 - Inject new process
 - Inject Dynamic Link Library (DLL)
 - Inject DLL 2 (Reflective)



Virtual address descriptor (VAD)

- Windows organizes process memory in VADs
 - Set of memory chunks
 - Tree-like organized
 - Always updated
 - Chunk attributes
 - Start - End address
 - Read / Write / Execute
 -



Related Works

- Volatility
 - A great open-source memory forensic tool/framework
 - Written in Python
 - More forensic oriented
 - Only support memory-dump files
- Other works on Windows kernel internals
 - Thanks to great works done by various reversers giving excellent insights into Windows OS kernels!



Detecting rootkits

- Userspace rootkits
- Kernel rootkits



Project status

- Under heavy development
- Around 30.000 lines of code totally
 - Frameworks & toolset
 - Good support for Windows XP
- In-progress work
 - To support other edition of Windows
 - Non-resident memory access
 - Linux support



Toolset & code?

- Will be released very soon!
 - Please watch out for the announcement on mailing lists
 - Most likely under open source license



Acknowledgement

- Thanks to METI (Japan) for partly sponsoring this project in the "Next Generation Info-Security R&D" program.



Conclusions

- Put the Forensic and IR tool outside of protected VM has some significant advantages
 - Zero-cost on deployment
 - Never erase evidences
 - Invisible to malware
 - Tamper resistant to malware inside the VM
 - Can mitigate damages, or disable malware effectively from outside



Outspect: Live Memory Forensic & Incident Response for Virtual Machine

Q & A

Nguyen Anh Quynh

